



university of
groningen

faculty of science
and engineering

Colour Interpolation in Gradient Meshes

Bachelor's thesis

July 2017

Jonathan Hogervorst

Primary supervisor: Jiří Kosinka, PhD

Secondary supervisor: Pieter Barendrecht, MSc

Abstract

The gradient mesh tool is a useful tool for creating advanced colour gradients in vector images and is already available in common vector editing programs. A gradient mesh consists of a grid of cells, where colours can be assigned to vertices. The colours are interpolated within the cells, yielding colour in the whole mesh. However, current implementations offer little control over the method used for colour interpolation.

This project involves the implementation and evaluation of various advanced colour interpolation settings in an existing gradient mesh tool. Several colour spaces and interpolation functions are implemented, allowing easy switching between them. Additionally, a feature is implemented that allows assigning colours to derivatives. Finally, a user study is performed to determine the usability of the tool and user's preferences regarding interpolation settings.

The results suggest that the CIELUV colour space and the cubic interpolation function are the best choices for colour interpolation. Nevertheless, depending on artistic style, other interpolation functions may still be useful. Assigning colours to derivatives does not yield the effect users expect and is thus no meaningful feature. However, users liked the extra control through derivatives, so other approaches should be investigated.

Contents

1	Introduction	3
2	Background	5
2.1	Gradient mesh	5
2.2	Colour spaces	5
2.2.1	CIE colour spaces	6
2.2.2	RGB colour spaces	6
2.2.3	Cylindrical colour spaces	7
2.3	Interpolation and interpolation functions	7
2.4	Colour interpolation	7
2.5	Continuity	8
2.6	Existing tool	8
3	Implementation	9
3.1	Colour spaces	9
3.2	Interpolation functions	10
3.3	Assigning colours to derivatives	11
3.4	Additional improvements	11
3.4.1	GUI improvements	11
3.4.2	Multithreaded colour interpolation	12
3.4.3	Uniform colour indexing tool	12
4	Results and evaluation	14
4.1	Colour spaces	14
4.2	Interpolation functions	14
4.3	Assigning colours to derivatives	14
4.4	User study	17
5	Discussion	23
5.1	Conclusion	23
5.2	Reflection on the user study	24
5.3	Future work	24

1 Introduction

Vector images provide several advantages over raster images: they can be scaled without loss of quality, the file size is often smaller, and certain editing operations can be done better. However, it is difficult to produce photorealistic vector images based on real-world scenes, since you need to get colours and light exactly right. One of the tools that are used for designing vector images is the gradient mesh. It consists of a grid of cells, where colours can be assigned to vertices. Colour interpolation is used to colour the whole gradient mesh, yielding colour gradients between the vertices. The gradient mesh tool is already available in common vector editors like Adobe Illustrator, CorelDRAW, and Inkscape.

Current gradient mesh implementations offer few user-customizable settings regarding colour interpolation. In most implementations, it is possible to assign colours at control points, move control points, and move tangent handles. In Inkscape it is also possible to choose between bilinear and bicubic interpolation. However, more advanced settings are not available.

Related work There has been some previous work on gradient meshes. Most of it was aimed at generating or colouring gradient meshes in some way: either automatic mesh generation from raster images [6, 12]; colour transfer from an example image to an existing gradient mesh [14]; or colouring an existing gradient mesh by scribbling [13]. Other research was done into making gradient mesh structures more flexible using cubic mean value coordinates [7] or subdivision [8]. Finally, there has been some research into local refinement of gradient meshes [9].

Contributions This project involves the implementation and evaluation of various advanced colour interpolation settings in an existing gradient mesh tool.

First of all, several colour spaces used for colour interpolation are compared. To easily view and compare the result of colour interpolation in different colour spaces, it is possible to choose between several colour spaces in the tool. Secondly, several interpolation functions that can be used for colour interpolation are compared. To compare the result of colour interpolation using different interpolation functions, a dynamic colour interpolation function was implemented. This allows choosing between interpolation functions in the tool (linear, cubic, or something else). Thirdly, the usefulness of assigning colours to gradients and mixed-partial derivatives is investigated. For this, UI controls were implemented to allow assigning colours to derivatives at control points in the gradient mesh.

Finally, I performed an informal user study to determine the usability of the tool and user's preferences regarding the interpolation settings. Based on the visual results of the tool and the results of the user study, I determined which colour space and interpolation function should be used for colour interpolation. Additionally, I determined the influence and usefulness of assigning colours to gradients and mixed-partial derivatives.

Research questions The research questions of this project are as follows:

- Which colour space should be used for colour interpolation (sRGB, linear sRGB, CIELAB, or yet another space)?

- Should colour interpolation be (bi)linear, (bi)cubic, or something else still?
- Colour in the mesh is traditionally assigned to vertices only. Can we meaningfully assign colours to the gradients as well?
- What influence do mixed-partial derivatives have on the resulting colour gradient? And how can this extra freedom be presented to the user?

Outline In Section 2, I provide background information regarding gradient meshes, relevant properties of colour interpolation, and the existing tool. In Section 3, I describe the implementation of the colour interpolation settings and some additional improvements to the tool. In Section 4, I provide my results and evaluation, using the visual results of the tool and the results of the user study. Finally, in Section 5, I provide my conclusions, a reflection on the user study, and ideas on future work.

2 Background

2.1 Gradient mesh

The common gradient mesh implementation was defined in [12] as a grid consisting of Ferguson patches [3]. This method was applied in [6, 14], as well as in the existing tool used for this project [9]. A Ferguson patch is enclosed by four boundary curves, which can be defined by Bézier curves. The patch is defined by its four vertices and the first- and second-order derivatives at each vertex. In a Ferguson patch, the second-order derivatives are set to zero. Non-zero second-derivatives are allowed in the similar Coons patch [2]. An example of a Ferguson patch is shown in Figure 1. As described in [9], the patches in the existing tool are evaluated using

$$F(u, v) = U(u) C M C^T U^T(v), \quad (1)$$

where

$$U(u) = \begin{bmatrix} 1 & u & u^2 & u^3 \end{bmatrix},$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{bmatrix}, \text{ and } M = \begin{bmatrix} m^0 & m^2 & m_v^0 & m_v^2 \\ m^1 & m^3 & m_v^1 & m_v^3 \\ m_u^0 & m_u^2 & m_{uv}^0 & m_{uv}^2 \\ m_u^1 & m_u^3 & m_{uv}^1 & m_{uv}^3 \end{bmatrix}.$$

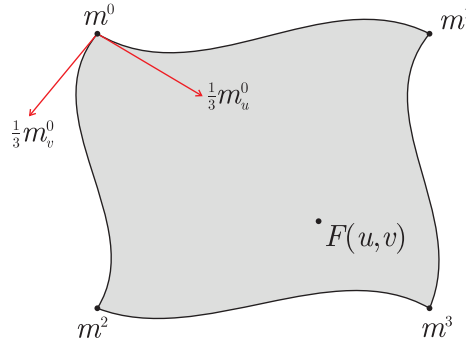


Figure 1: A Ferguson patch with boundary curves, control points, and the first-order derivatives at control point m^0 (adapted from [9])

2.2 Colour spaces

Colours exist due to differences in the wavelength of light transmitted by objects. To use colours in computer programs, there must be some representation of colour. Colour spaces provide such a representation. E.g., an RGB colour space defines colours based on additive numerical red, green, and blue values. A large number of colour spaces exist, each having their own properties. Most colour spaces use three channels. A colour is represented in such a colour space by three numerical values, one for each channel.

A description of some colour spaces is given below. Two important properties of colour spaces are linearity and perceptual uniformity. In a linear colour

space, colours can be scaled linearly. In a perceptually uniform colour space, the Euclidian distance between two colours is proportional to the difference between the colours as perceived by humans.

2.2.1 CIE colour spaces

CIE 1931 colour spaces The CIE 1931 XYZ (**CIEXYZ**) colour space was designed in such a way that it could reproduce all colours visible to humans. Studies into the human vision system revealed that the human eye uses three kinds of cones, responsive to different light wavelengths. The CIEXYZ colour space defines tristimulus values for the X , Y , and Z component, which can be thought of as primary colours in the context of the CIEXYZ colour space. The values are not exactly the same as the wavelengths perceived by human eye cones, nor are they real colours. However, using these tristimulus values, all colours that are visible to humans can be represented in the CIEXYZ colour space using positive values only. The CIEXYZ colour space is a linear colour space, but it is not perceptually uniform.

Another CIE 1931 colour space is the CIE 1931 RGB colour space. This colour space also provides three tristimulus values for the colour components. More information about RGB colour spaces can be found below, though the specific CIE 1931 RGB colour space will not be considered further.

CIE 1976 colour spaces The CIE 1976 $L^*a^*b^*$ (**CIELAB**) and CIE 1976 $L^*u^*v^*$ (**CIELUV**) colour spaces were designed to reproduce all perceivable colours and provide perceptual uniformity. As a result of providing perceptual uniformity, converting a colour from CIEXYZ to CIELAB or CIELUV cannot be done linearly, which makes the CIELAB and CIELUV colour spaces nonlinear. The L^* component specified the lightness of a colour. In the CIELAB colour space, the a^* component specifies the position between magenta and green, and the b^* component specified the position between yellow and blue. In the CIELUV colour space, a different representation is used.

2.2.2 RGB colour spaces

RGB, or red-green-blue, colour spaces are also based on the human perceptive system. Similar to the CIEXYZ colour space, RGB colour spaces use three tristimulus values for the colour components. Various RGB colour spaces exist, each using different values. Contrary to the CIEXYZ colour space, not all human perceivable colours can be represented in RGB colour spaces.

sRGB sRGB is the most commonly used colour space for computers, displays, scanners, and the Internet. The sRGB standard defines tristimulus values for red, green, and blue, the white point, and gamma correction. To convert a colour from CIEXYZ to sRGB, a linear transformation and (a nonlinear) gamma correction are applied on the CIEXYZ colour values. The gamma correction is applied to correct for the fact that the human eye perceives light nonlinearly.

Due to the nonlinear gamma correction, the sRGB colour space is no linear colour space. Additionally, it is also not a perceptually uniform colour space.

Linear sRGB Linear sRGB is a variant of sRGB which does not include gamma correction. As such, a colour in CIEXYZ can be converted to linear sRGB using just a linear transformation, which makes linear sRGB a linear colour space. It is still not a perceptually uniform colour space.

2.2.3 Cylindrical colour spaces

Cylindrical colour spaces provide a different representation than, e.g., CIEXYZ or RGB colour spaces. In a cylindrical colour space, the hue of a colour is encoded in a single channel as the angle in a cylinder. In contrast, in other colour spaces, the hue is encoded in two (CIELAB and CIELUV) or three channels (CIEXYZ and RGB). Due to the cylindrical representation of hues, these colour spaces are not linear.

HSV/HSL/HSI Common cylindrical colour spaces are HSV, HSL, and HSI. In these colour spaces, the H and S channel represent the hue and saturation of the colour. The third channel — V , L , or I — represents the value (brightness), lightness, or intensity, respectively. These colour spaces are not perceptually uniform.

CIELCh There also exist cylindrical versions of CIELAB and CIELUV, which are known as **CIELCh_{ab}** and **CIELCh_{uv}**. In these representations, the hue is encoded in the h channel, while the chroma is encoded in the C channel. The lightness, encoded in the L channel, is the same as in the CIELAB and CIELUV colour spaces. The CIELCh colour spaces are still perceptually uniform.

2.3 Interpolation and interpolation functions

The problem of interpolation is: given two (or more) points, how to find appropriate points in between? This problem is solved by finding a function which passes through the given points. This function can then be used to find additional points in between.

A simple approach is the use of a linear interpolation function. In this case, the interpolation function has a fixed slope, and thus goes with constant speed from one point to another. A more advanced approach is the use of a polynomial function. In this case, the interpolation function is a polynomial of some degree. A specific case is cubic interpolation, using a polynomial of degree three.

2.4 Colour interpolation

The problem of colour interpolation is: given two colour endpoints, how to find an appropriate colour gradient in between? This problem is actually twofold: first, the colours must be represented numerically, probably using some colour space; and second, interpolation must be done using some interpolation function.

The colour spaces that are considered in this project all use three channels. The common method of colour interpolation between two such colours is to apply the interpolation function separately to each of the channels. Therefore, the result of the colour interpolation is affected by the colour space that is used as well as the interpolation function. Two example colour interpolations showing the differences between various colour spaces are shown in Figure 2.

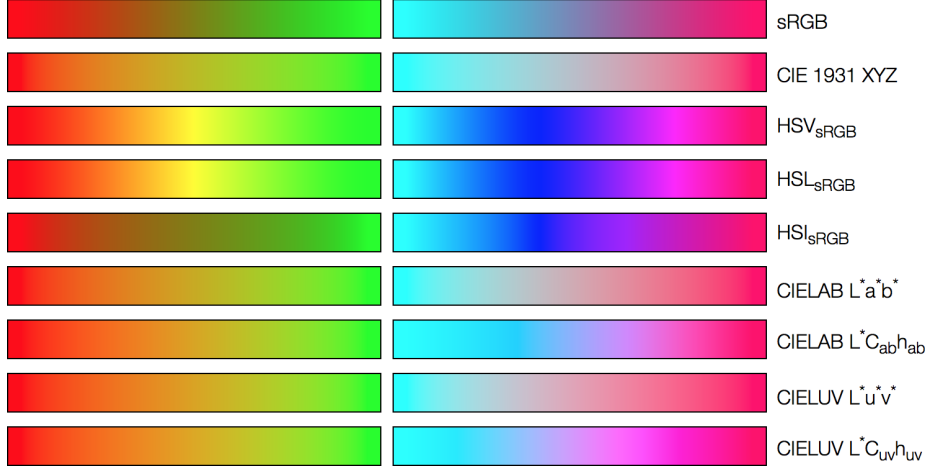


Figure 2: Linear colour interpolation in various colour spaces [10]. Left: red to green; right: cyan to pink.

2.5 Continuity

A relevant property of interpolation functions is the smoothness. The smoothness can be expressed by the continuity of the function or its derivatives. If a function is continuous, it means that its curve is connected at all times, i.e., it cannot jump from one value to another. Such a function is C^0 continuous. If the first derivative of the function is also continuous, we call the function C^1 continuous. The same applies for the second derivative (C^2 continuous), etc.

To achieve smooth transitions in the gradient mesh, geometry and colour should be continuous, both within and between patches. Linear and cubic interpolation functions yield C^∞ continuity within patches because the colour gradients and its derivatives are continuous in all directions. Therefore, to achieve continuity in the whole gradient mesh, only continuity between patches remains to be ensured.

2.6 Existing tool

This project builds upon an existing gradient mesh tool [9] with some improvements. The tool was built in C++ using the Qt framework and OpenGL. It supports the creation and manipulation of gradient meshes. Local refinement is possible by splitting patches in the mesh into four subpatches. The GUI allows users to move control points and (second) derivatives, and assign colours to control points. Colour interpolation is applied using a bicubic interpolation function in the CIELAB colour space.

The existing tool makes use of Coons patches for both geometry and colour interpolation. In the case of colour interpolation, the colours at the control points are converted to CIELAB and assigned to m^i in M , used in Equation (1). The derivatives are all set to zero, resulting in the current bicubic interpolation.

In addition to the version of the tool described in [9], some improvements have been made prior to the start of this project. This includes the ability to start with an $n \times m$ mesh.

3 Implementation

3.1 Colour spaces

To compare colour spaces, it must be possible to choose between several colour spaces to be used for colour interpolation. I implemented a number of radio buttons in the sidebar of the tool which can be used to switch between the colour spaces, as can be seen in Figure 3.

The colours assigned in the mesh are stored in the sRGB colour space. When performing the colour interpolation, the colours are first converted to the chosen colour space, then colour interpolation is applied, and finally, the interpolated colour values are converted back to sRGB to be displayed. To convert colours from sRGB to the chosen colour space and back, I use an existing colour conversion package [4].

The following colour spaces are available in the tool: CIEXYZ; CIELUV; CIELAB; sRGB; HSV; HSL; HSI; and CIELCh_{ab}. Linear RGB is not available as an explicit option: because it is a linear transformation from CIEXYZ, it yields the same interpolation result as CIEXYZ.

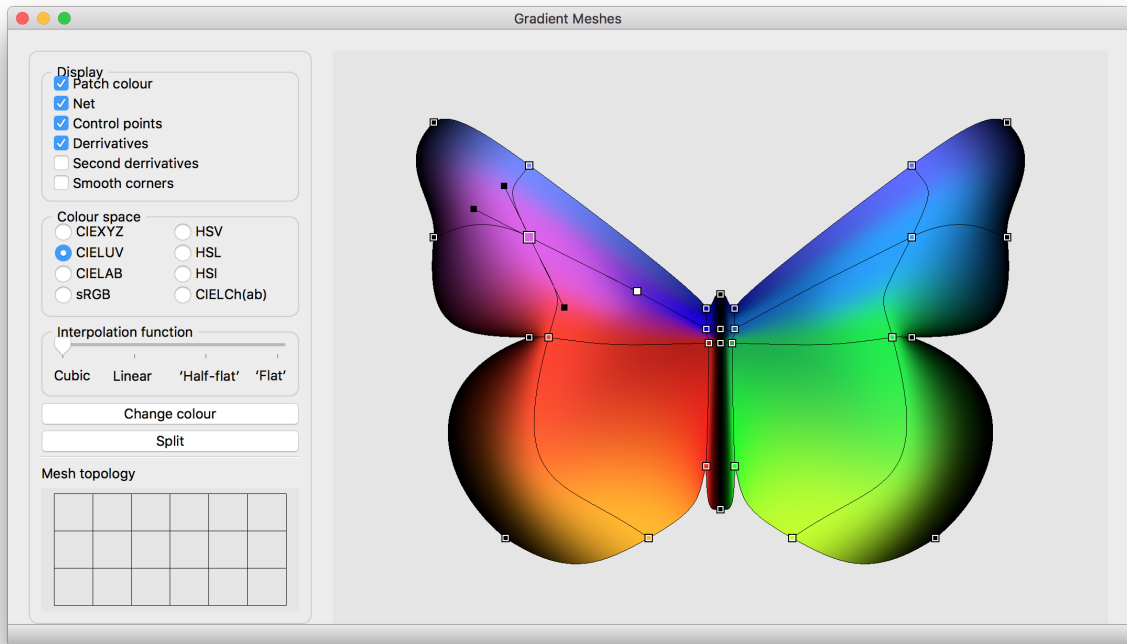


Figure 3: Extended gradient meshes tool. In the sidebar on the left, the colour space buttons and the interpolation function slider can be seen. Below and to the right of the selected pink control point, a derivative handle with an assigned colour (white) can be seen.

3.2 Interpolation functions

To compare interpolation functions, it must be possible to choose between interpolation functions to be used for colour interpolation. I implemented a slider in the sidebar of the tool which can be used to pick an interpolation function. The two ends of the slider correspond to the two extreme settings, cubic and ‘flat’. In-between those settings, other interpolation functions are available.

A visualisation of some interpolation functions in a univariate setting can be seen in Figure 4. The interpolation functions are visualised as Bézier curves. The first and last control points are fixed to the assigned colours, while the two middle control points are moved to vary between the different interpolation functions.

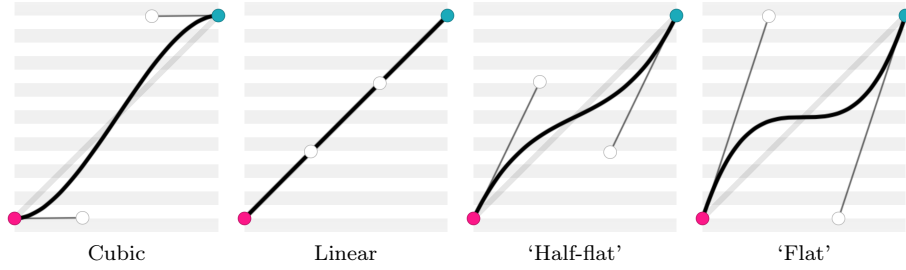


Figure 4: Some cubic interpolation functions in a univariate setting

In a bivariate setting, as in the patches in a gradient mesh, a similar approach is used, though the two directions must be taken into account. Based on the slider, the derivatives needed in M are calculated using

$$m_u^{\{0,1\}} = f \, 3 \, (m^1 - m^0), \quad m_u^{\{2,3\}} = f \, 3 \, (m^3 - m^2), \quad (2)$$

$$m_v^{\{0,2\}} = f \, 3 \, (m^2 - m^0), \quad m_v^{\{1,3\}} = f \, 3 \, (m^3 - m^1), \quad \text{and} \quad (3)$$

$$m_{uv}^{\{0,1,2,3\}} = f \, 9 \, (m^0 - m^1 - m^2 + m^3), \quad (4)$$

where f is the value of the slider, with $0 \leq f \leq 1$, and m^i are the colours at the control points. Position $f = 0$ of the slider corresponds to the cubic setting, $f = \frac{1}{3}$ with the linear setting, $f = \frac{2}{3}$ with the ‘half-flat’ setting, and $f = 1$ with the ‘flat’ setting. (N.b., while I call $f = 0$ the ‘cubic’ setting, all interpolation functions yielded by these derivatives are technically cubic because a polynomial of degree three is used.) After calculating the derivatives, colour interpolation is performed by evaluating the patch using Equation (1).

All interpolation functions are C^∞ continuous within a patch because the colour gradients and its derivatives are continuous in all directions. Additionally, all functions are at least C^0 continuous at the edges between patches: because overlapping control points of patches share the same colours, and the same interpolation function is applied on all patches in the mesh, the result will be the same on both sides of the edge, and thus continuous. However, only the cubic interpolation function is C^1 continuous: the derivatives are always zero, and thus continuous. With other interpolation functions, the derivatives may differ between patches and are thus not guaranteed to be continuous.

3.3 Assigning colours to derivatives

Besides the colour spaces and interpolation functions, I want to investigate the influence of assigning colours to derivatives. This comes down to letting the user assign colours of their choice to m_u^i and m_v^i , being the first-order derivatives, and m_{uv}^i , being the second-order derivatives. This is implemented by checking whether the user has assigned a colour to each derivative before performing the colour interpolation. If a colour was assigned, that value is used; if not, the value based on the interpolation function, calculated using Equations (2) to (4), is used.

Furthermore, we need to distinguish between derivatives with and without assigned colours in the GUI. In the existing tool, derivatives cannot have colours assigned to them and are always displayed in black. However, when just applying the assigned colour to the derivative handle, it would not be possible to distinguish between derivatives without colour and derivatives with the colour black. To improve this, we show a white and black border around derivatives, indicating that the colour in the middle is an assigned colour. For consistency, this approach is also used for control points. Finally, to more easily distinguish between first-order and second-order derivatives, we use grey instead of black for the handles and borders of second-order derivatives.

A screenshot of the extended gradient meshes tool can be seen in Figure 3.

3.4 Additional improvements

3.4.1 GUI improvements

Assigning colours To enable users to easily change the colour at a control point, a feature was implemented to allow middle-clicking on control points to change their colour. Previously, users were required to first select (left-click) the control point, and then click the ‘Change colour’ button in the sidebar.

Additionally, a check was implemented to only assign colours from the colour picker if the user chose a colour. Previously, when opening the colour picker and cancelling, an invalid colour was assigned to that control point. After this improvement, the colour of the control point is not changed when cancelling the colour picker.

Custom mesh dimensions To enable users to easily create meshes with arbitrary size, a dialogue box was implemented that allows users to specify the width and height of the mesh. When starting the program, a 2×2 mesh is initialized. However, when clicking ‘Reset mesh’ in the menu, the new dialogue box is shown, allowing users to create an $n \times m$ mesh. Previously, the mesh dimensions were hard-coded in the source code, requiring users to change the source code and recompile the program to use a different mesh size.

Zooming improvements In the existing tool, the mesh can be zoomed by scrolling. However, scrolling did not work properly on macOS, and scrolling using a trackpad did not work properly on Ubuntu. Therefore, the function responsible for updating the zoom level in response to scroll events was changed, fixing these issues.

3.4.2 Multithreaded colour interpolation

When running the existing tool on my development machine¹, some performance issues showed up. When modifying the gradient mesh, e.g., by dragging a control point, the tool was not able to smoothly keep up with the dragging movement. The Activity Monitor application from macOS revealed that the program used 100% of a single CPU core while dragging a control point. My hypothesis was that calculating the updates during dragging required more than 100% of a CPU core on this machine. Because that is not possible in a single thread, dragging became sluggish.

Further inspection using the Time Profiler of the Apple developer Instruments application revealed that 45% of the CPU time during usage of the tool was spent on the colour interpolation in the gradient mesh. As a trial, I decided to parallelize the colour interpolation. Using the Qt Concurrent API [11], it was possible to perform the colour interpolation for each root patch in the gradient mesh in parallel, without drastically changing the structure of the program.

After this modification, the tool used 140% CPU while dragging. However, this activity was divided over multiple threads, none of which were using the whole capacity of a single CPU core. This confirmed my hypothesis that the calculations were too heavy for a single core on this machine. Furthermore, this parallelization solved the performance issues.

3.4.3 Uniform colour indexing tool

To evaluate colour interpolation results, it can be useful to convert resulting images into indexed-colour images. In indexed-colour images, a limited number of colours is used. When converting an image, each pixel gets the closest colour from the palette of available colours. When using a palette with evenly-spaced colours, a colour gradient will be split uniformly into parts with the same colour, making the progression of the gradient better visible to the human eye.

Raster image editors like GIMP and Adobe Photoshop include a feature to convert an image to indexed-colour mode. However, Adobe Photoshop supports at most 6 colour levels per RGB channel in the ‘Uniform’ palette type [1], while GIMP does not support uniform palettes at all [5].

Therefore, I decided to create a simple tool to apply indexed-colour mode using evenly-spaced colour palettes. For cross-platform compatibility, the tool was implemented as a simple single-page web application, using the HTML5 canvas element and corresponding JavaScript API for client-side image manipulation and the Bootstrap framework for the GUI. Users can pick an image file in a format supported by their browser. Furthermore, they can choose how many levels per colour channel must be used. Additionally, users can choose whether the alpha channel of the input image must be removed, or whether the input image should be converted to greyscale prior to indexing. The resulting indexed-colour mode output image is generated immediately and can be viewed in the browser or saved as PNG file. A screenshot of the tool can be seen in Figure 5. The tool can be used at <https://jhogervorst.github.io/indexed-colour-image/>.

¹MacBook Pro with an Intel Core i7-3740QM CPU @ 2.70GHz

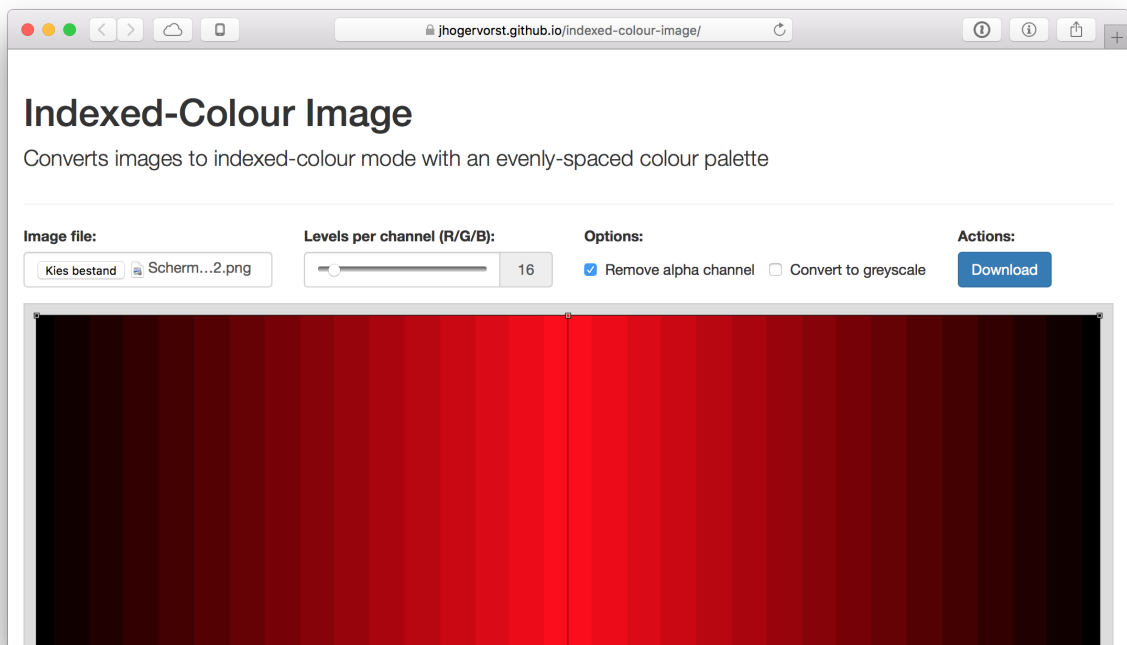


Figure 5: Uniform colour indexing tool

4 Results and evaluation

4.1 Colour spaces

In Figures 6 and 7, example images are shown using the different colour spaces available in the tool.

The first thing to notice is that the cylindrical colour spaces (HSV, HSL, HSI, and CIELCh_{ab}) show unnatural interpolation results. Especially in the transition from red to magenta, many different hues are visible. This can be explained due to the fact that these cylindrical colour spaces use only one channel for the hue. In the H channel in HSV, HSL, and HSI, red ($H = 0^\circ$) is one of the extremes, and magenta ($H = 300^\circ$) is near the other extreme, since $0^\circ < H < 360^\circ$. As a result, the colour interpolation goes through all almost all possible values of H , yielding a rainbow of different hues.

The four other colour spaces, CIEXYZ, CIELUV, CIELAB, and sRGB yield similar results. The biggest difference is the darkness in the colour transitions. CIEXYZ is relatively light, while CIELUV and CIELAB are similar and darker, and sRGB is much darker. This difference can be explained due to the perceptual uniformity of CIELUV and CIELAB, and the gamma correction applied in sRGB.

Both CIELUV and CIELAB were designed to be perceptually uniform, making them theoretically especially suitable for colour interpolation. One minor difference is visible between the two: in CIELAB, the transition from blue to green goes through purple, which is not natural. This does not happen in CIELUV.

4.2 Interpolation functions

In Figures 8 and 9, example images are shown using some of the interpolation functions available in the tool. In Figure 10, example images are shown alongside their indexed-colour mode variants.

The ‘half-flat’ and ‘flat’ interpolation functions yield large single-coloured blobs in the middle of patches. While this may be desired for certain artistic styles, it is generally not the result one would expect in a gradient mesh.

When comparing the other interpolations functions, cubic and linear, the biggest difference can be seen near the vertices in the mesh. In the cubic setting, the colour assigned to the vertex is clearly visible around it. In the linear setting, you see an immediate transition towards the colour at the opposite vertex. It is as if, in the cubic setting, more focus is put on the colours at the vertices, instead of the transition in-between vertices. The cubic setting seems like the better option: colours are assigned to vertices for a reason, so there should be some focus on them. Furthermore, the cubic interpolation function provides C^1 continuity between patches.

4.3 Assigning colours to derivatives

In Figure 11, example images are shown with colours assigned to derivatives. The cubic interpolation function is used, which uses zero derivatives by default. This should yield the biggest effect when assigning colours to derivatives.

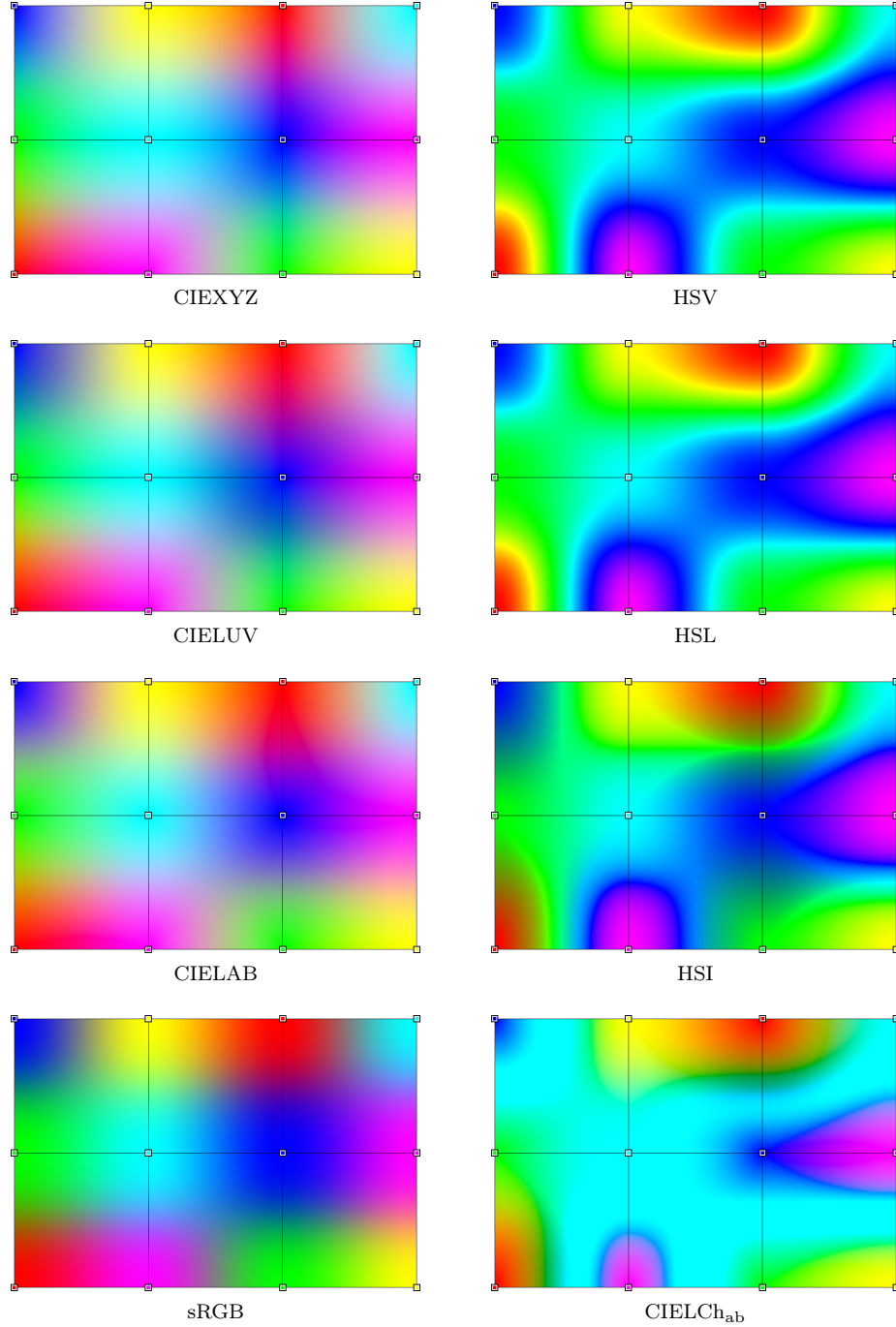


Figure 6: Colour interpolation results using different colour spaces with the cubic interpolation function. All common primary colours are laid out such that they all have a ‘direct’ interpolation (along an edge) with each other.

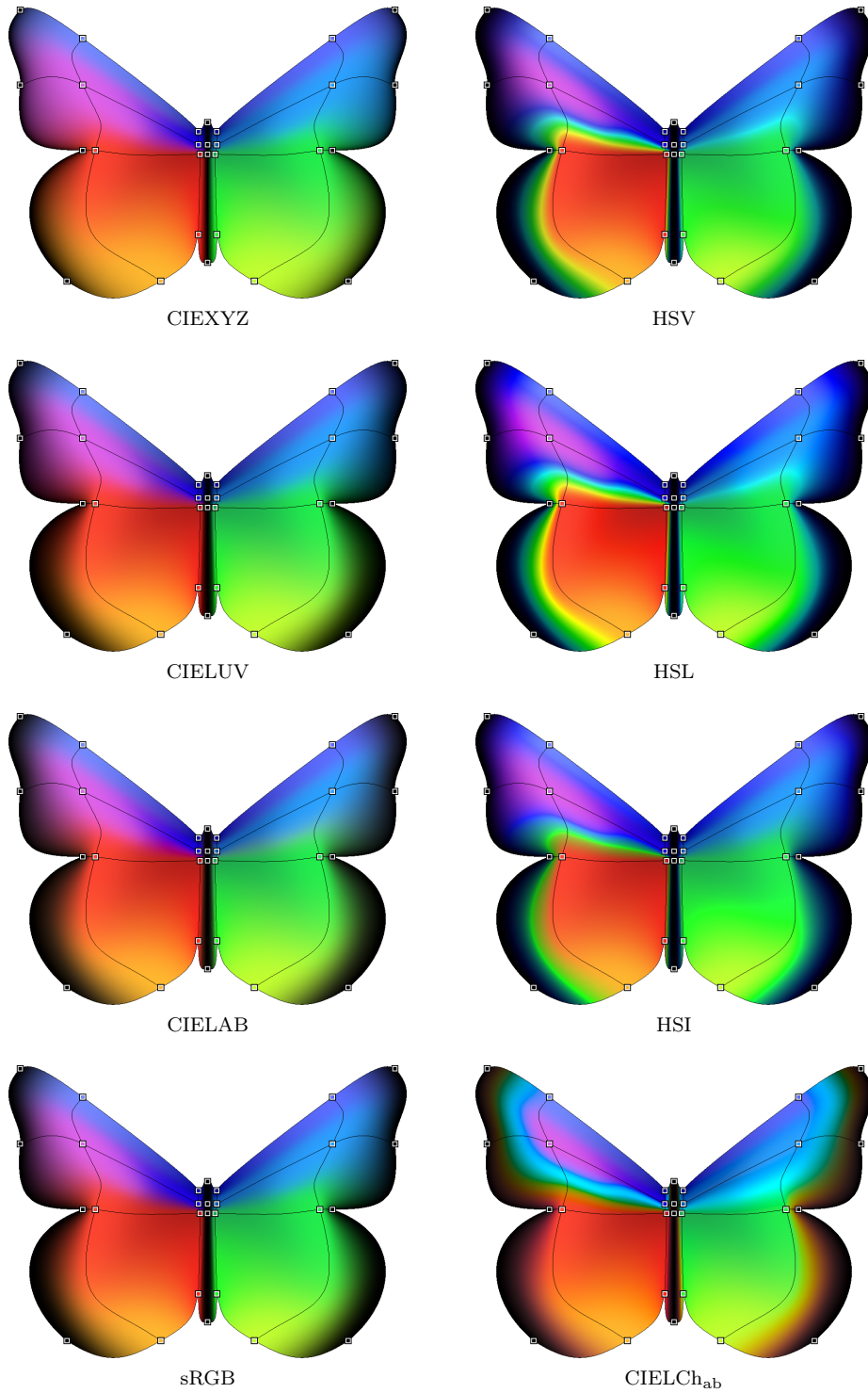


Figure 7: Colour interpolation results on an example image using different colour spaces with the cubic interpolation function

In the second row of images, blue was assigned to one of the first-order derivatives of the selected control point. There is a slightly visible change, though blue itself cannot be seen in the colour gradient. In the third row of images, yellow was assigned to both first-order derivatives of the selected control point. The effect is better visible now, though it depends on the colour space used: in the CIELUV colour space, the yellow colour is best visible. In the fourth row of images, yellow was assigned to the second-order derivative of the selected control point. This effect is hardly visible.

4.4 User study

To evaluate the extended gradient meshes tool, an informal user study was performed. Participants were given a short introduction about the tool and were asked to try things and express their remarks. During the trials, participants were asked to perform common operations, like moving control points and (second) derivatives, assigning colours to control points and (second) derivatives, and changing the colour space and interpolation function. Seven students from the Computing Science and Artificial Intelligence programs at the University of Groningen participated. They were given a small compensation (some food) in exchange for their time.

The remarks and suggestions from the participants are summarised below:

Colour spaces

- One participant preferred the CIELUV and CIELAB colour spaces. Another preferred the CIEXYZ colour space.
- One participant thought that cylindrical colour spaces could be useful to make certain images.
- One participant asked why the CMYK colour space was not included.
- One participant suggested using a colour picker based on the active colour space for assigning colours. (E.g., an RGB colour picker if sRGB is active, an HSV colour picker if HSV is active, etc.)

Interpolation functions

- Some participants liked the slider for dynamically selecting interpolation functions.
- Some participants wanted to modify the colour space and interpolation function after splitting, which is not possible due to the current implementation.

Assigning colours to derivatives

- Some participants did not expect the effect when assigning colours to (second) derivatives since the assigned colours do not become clearly visible.
- One participant liked the feature of assigning colours to (second) derivatives. Yet another thought that it introduced too many options, and would prefer splitting to influence colours in gradients.

- Some participants thought that the effect of assigning colours to (second) derivatives was too small. Some suggested to increase the effect, or allow the user to change the intensity of the effect.
- One participant found the black colour of derivatives confusing, since it is similar to derivatives with a black colour assigned, and suggested to use different shapes to distinguish between the two situations.

Derivatives

- One participant liked the feature of controlling second derivatives, though they found it difficult to understand.
- One participant suggested displaying the line between a control point and a (second) derivative in the colour assigned to the (second) derivative, to make a visual suggestion of the transition between the colours.
- One participant suggested to display the second derivatives slightly offset into the ‘right’ direction by default, so they are not all stacked onto the control point.

Influencing colour

- One participant expected to be able to change colours at all vertices after splitting.
- One participant suggested allowing adding control points on edges.
- One participant suggested allowing moving the ‘colour’ control points, without moving the ‘position’ control points.
- One participant asked how sharp lines could be made.

Gradient mesh

- One participant found the mesh lines and lines between control points and derivatives too dark if the colours behind were dark, and suggested to make them white on dark backgrounds.
- Several participants found the ‘folding’ effect when dragging a control point near or into another patch, strange. One participant asked whether the gradient mesh was 3D, after seeing this 3D-like effect.
- One participant suggested allowing creating 3D meshes.

GUI

- (Almost) all participants asked for an undo feature.
- Two participants found it difficult to mentally translate between the topology widget and the actual gradient mesh. One participant suggested moving the ‘Split’ button to the topology widget, to make their relationship more clear. Another participant suggested allowing zooming in the topology widget.
- One participant suggested creating info tooltips for the various options.

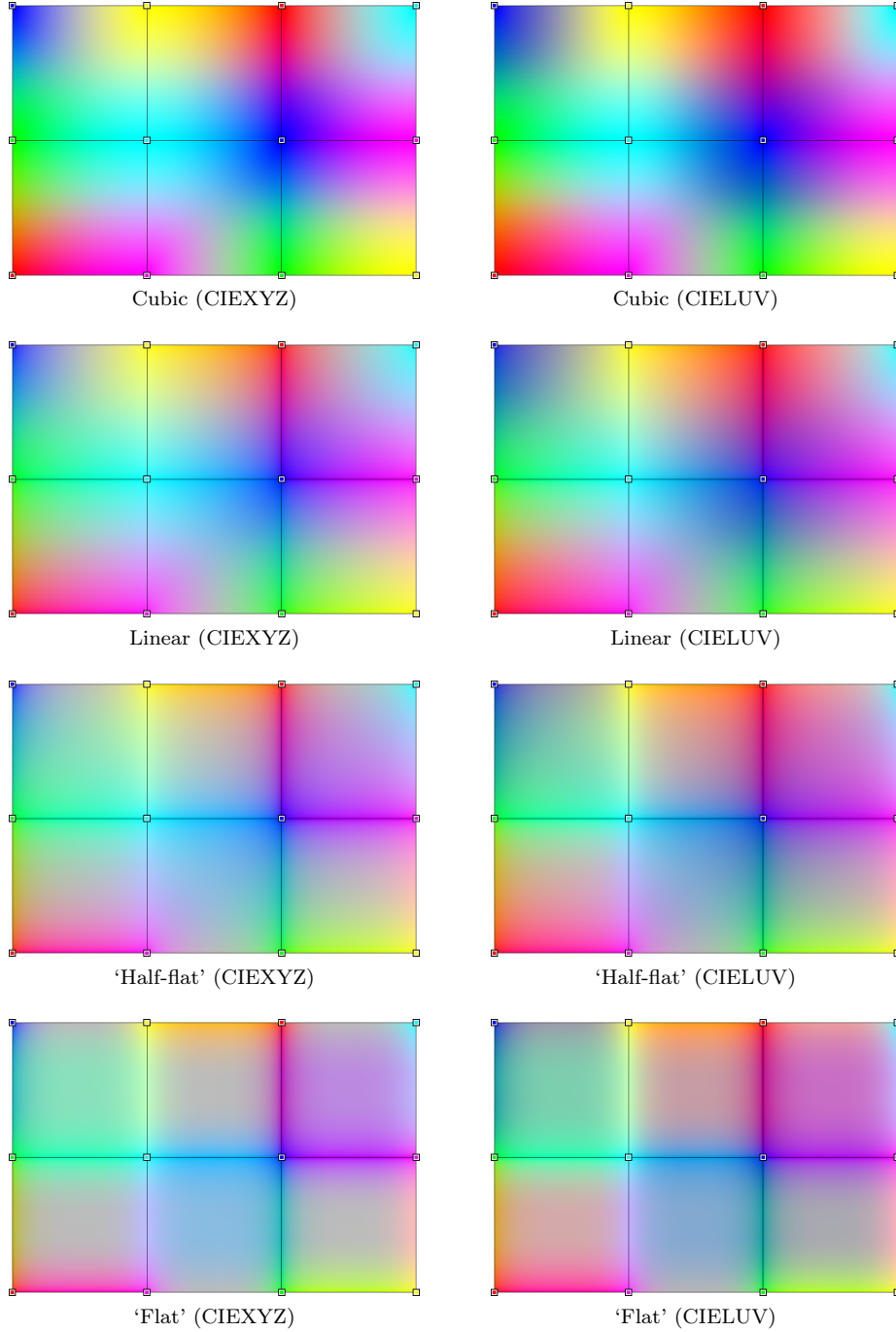


Figure 8: Colour interpolation results using different interpolation functions. All common primary colours are laid out such that they all have a ‘direct’ interpolation (along an edge) with each other.

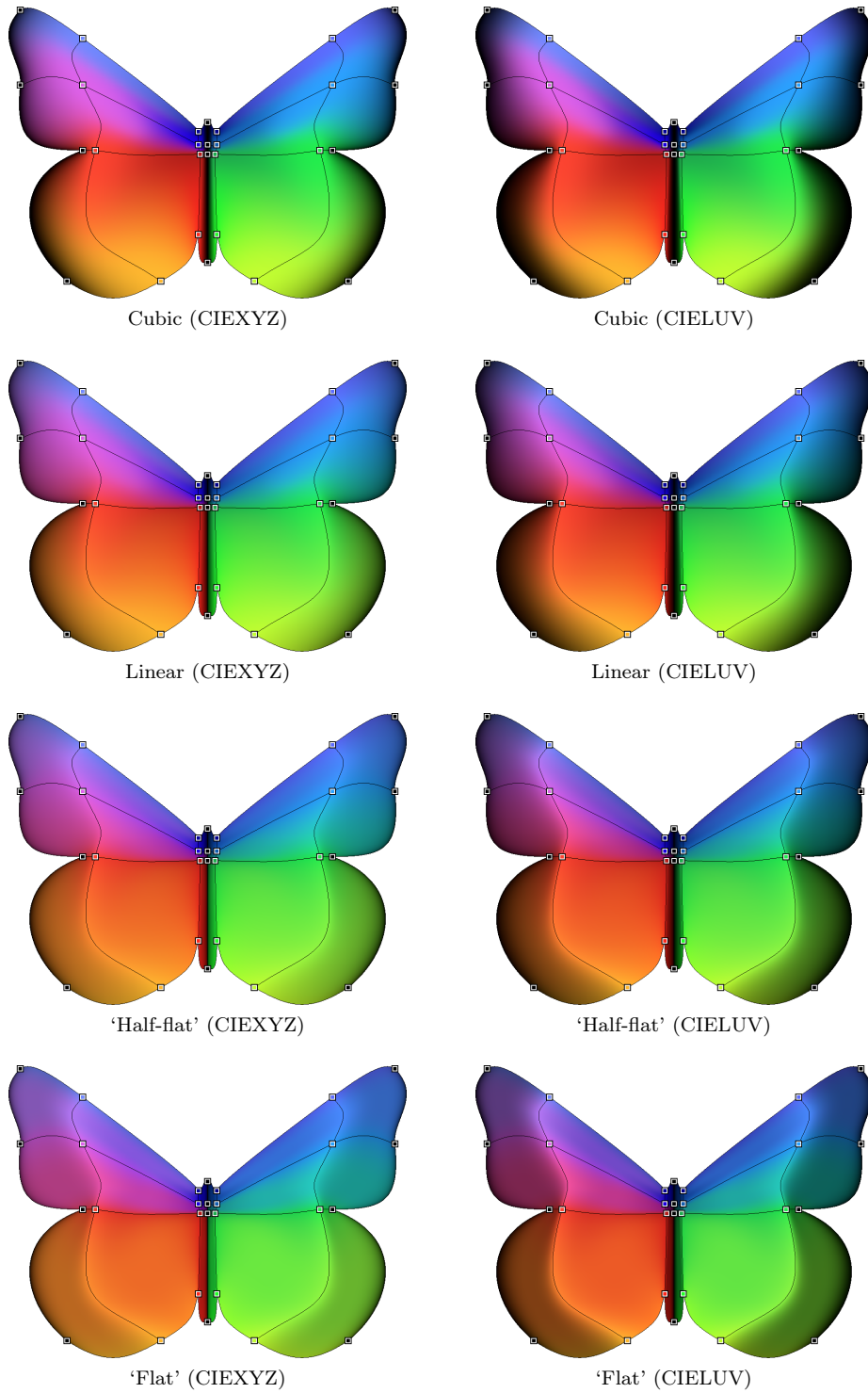


Figure 9: Colour interpolation results on an example image using different interpolation functions

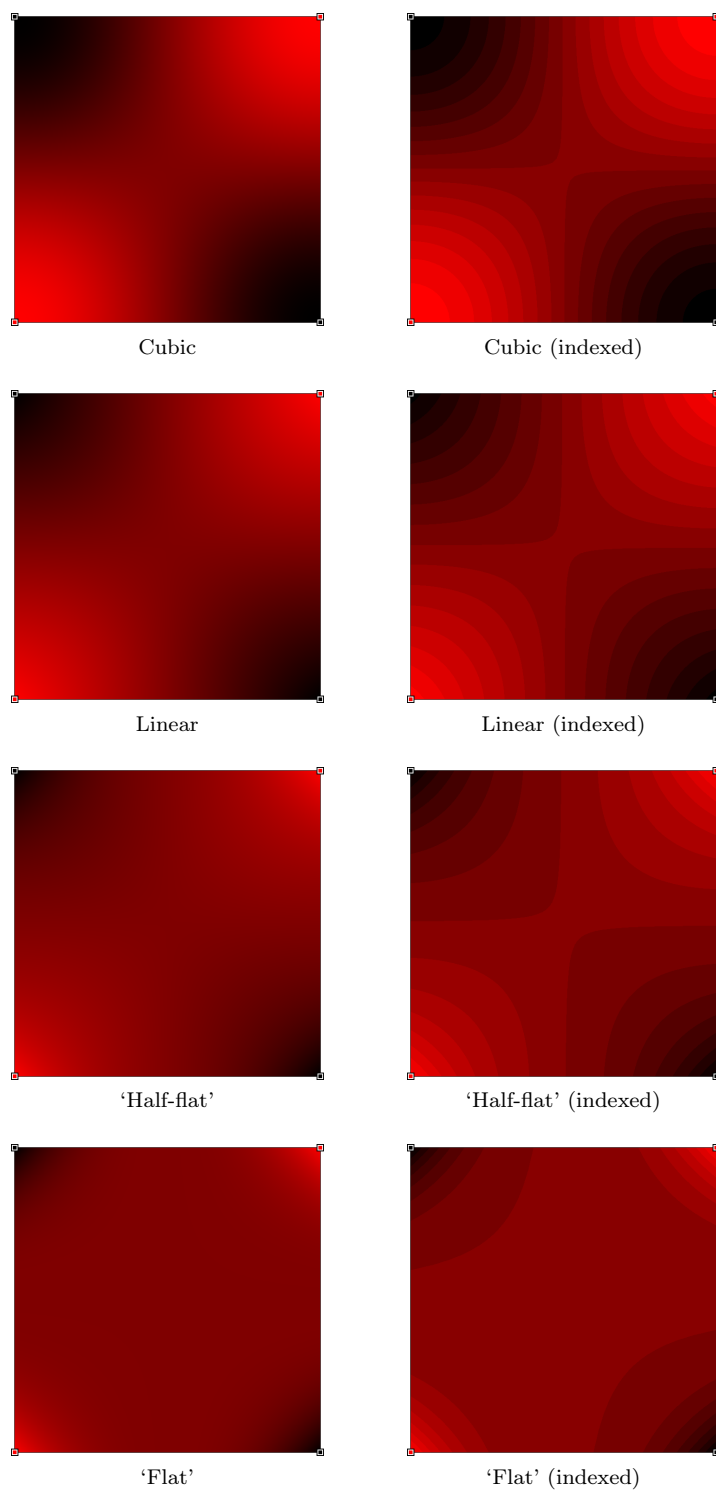


Figure 10: Colour interpolation results using different interpolation functions in the sRGB colour space. Indexed-colour mode images using 8 levels per channel.

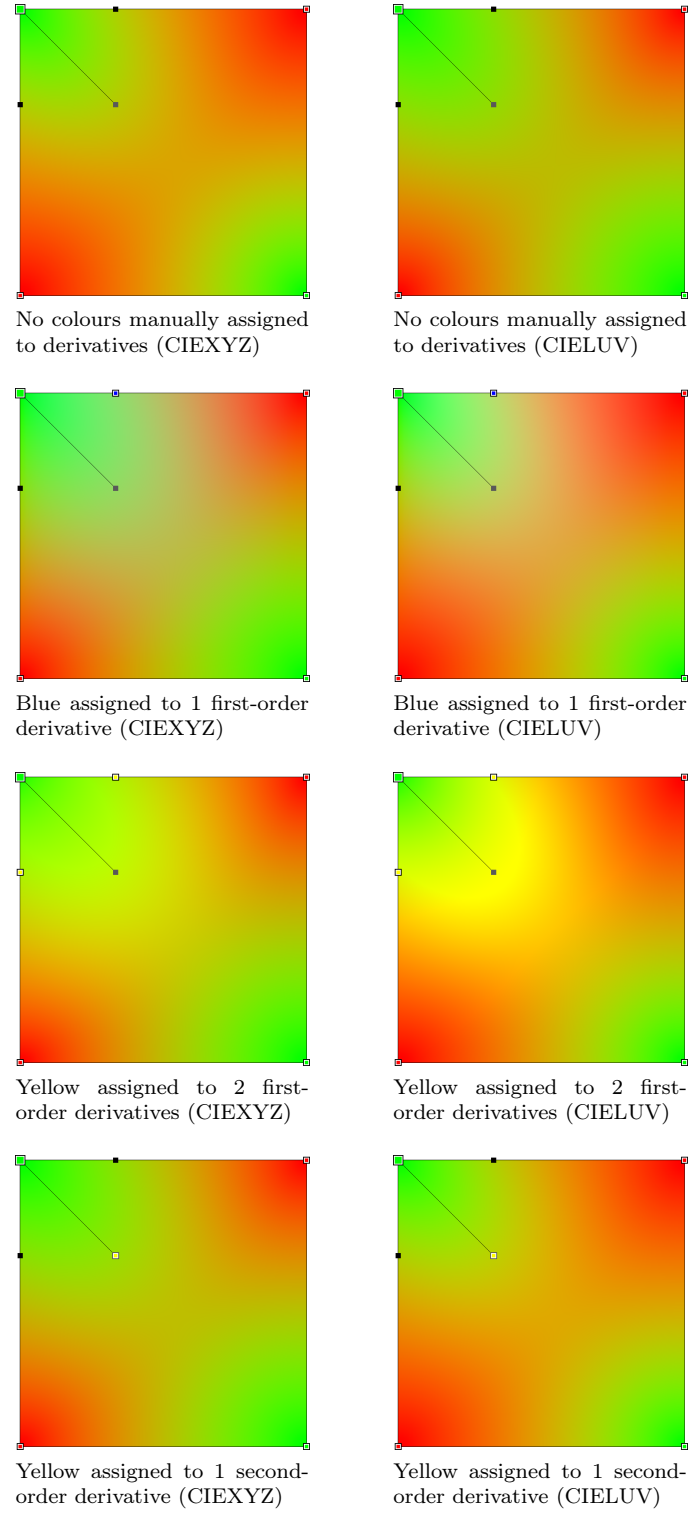


Figure 11: Colour interpolation results with colours assigned to derivatives using the cubic interpolation function

5 Discussion

5.1 Conclusion

The changes required to evaluate colour interpolation results and answer the research questions were implemented successfully. Additionally, a tool was created to convert images to indexed-colour mode, to easily analyse colour interpolation results. Finally, the informal user study yielded valuable feedback.

Which colour space should be used for colour interpolation (sRGB, linear sRGB, CIELAB, or yet another space)?

CIELUV seems to be the best colour space for colour interpolation. Cylindrical colour spaces yield very unnatural results and are thus not useful. CIEXYZ and sRGB are not perceptually uniform, which makes them less suitable for colour interpolation. Additionally, they provide relatively light or dark results, respectively, which may not be desired. CIELAB and CIELUV both seem suitable for colour interpolation. Because the former shows some minor undesired effects, CIELUV seems to be the best choice.

A downside of CIELUV (or CIELAB) is the lack of support in graphics libraries like OpenGL, which often only support (linear) sRGB. Without native support, performing colour interpolation on the GPU in CIELUV is much more expensive, since manual conversions are required. Therefore, sRGB may still be preferred for colour interpolation in high-performance applications.

Should colour interpolation be (bi)linear, (bi)cubic, or something else still?

The cubic interpolation function seems to be the best for colour interpolation. The ‘half-flat’ and ‘flat’ interpolation functions yield results one would not expect in a gradient mesh. The linear interpolation function is more suitable, though it rarely shows the colours at the vertices. Because these colours are assigned by the user, it seems logical to show them more prominently. Furthermore, the cubic interpolation function provides C^1 continuity, both within and between patches. Altogether, this makes it the best choice.

However, depending on artistic style, other interpolation functions may still be desired in certain situations. The interpolation function slider may be a good approach, as users in the user study liked it.

Colour in the mesh is traditionally assigned to vertices only. Can we meaningfully assign colours to the gradients as well?

As shown by the results, assigning colours to first-order derivatives clearly has an effect. However, the effect is not consistent, since it depends on the colours that are used. While there are visible changes, they may not be what a user would expect. This was also confirmed by the user study. Therefore, it seems like assigning colours to first-order derivatives is no meaningful feature.

Nevertheless, it may be useful to provide some control over the first-order derivatives, since they can have a visible influence on the colour interpolation. In the user study, users expressed that they liked to have some control over the derivatives. Assigning colours does not provide the expected results, but a different method may be used to let users influence first-order derivatives in colour interpolation.

What influence do mixed-partial derivatives have on the resulting colour gradient? And how can this extra freedom be presented to the user?

As shown by the results, assigning colours to second-order derivatives has a very small effect. Similar to first-order gradients, assigning colours to second-order derivatives seems no meaningful feature. Maybe a different method of influencing second-order derivatives in terms of colour can be more useful.

In addition to assigning colours, it is also possible to manipulate the colour gradient by moving the second-order derivatives. In the user study, users liked the feature, though they found it difficult to understand. Maybe this feature should be an advanced feature that can be enabled when users need it and is disabled by default.

5.2 Reflection on the user study

The user study helped to gain insight into people using the gradient mesh tool for the first time. Participants got a short introduction to the tool and were then allowed to use it themselves. They gave useful feedback in terms of the GUI and usability of the tool itself. While most of this was not useful for answering the research questions of this project, it can be useful to improve the tool in the future.

The user study was informal, meaning that participants were free to try what they wanted. Some suggestions were given to make sure that all participants used the most important features related to this project. However, they all approached it differently: some tried and compared all interpolation options; others tried to design something nice; and some tried to find bugs by performing strange actions. To get more specific information, like participant's opinions on the results of colour spaces and interpolation functions, a more strict user study could have been better, asking users to compare specific settings.

Furthermore, the user study was performed with students from the Computing Science and Artificial Intelligence programs, whom I knew personally. Some had technical knowledge about colour spaces or other related concepts. However, most were not experienced designers, while the tool is aimed at designers. Therefore, it might have been better to also include designers into the user study, since they might approach things differently.

5.3 Future work

Most of the suggestions for future work from the existing tool in [9] still apply. My additional suggestions are as follows:

- **Evaluating colour spaces and interpolation functions using more images:** In this research, one example image of a butterfly was used to evaluate the colour spaces and interpolation functions, in addition to some more abstract testing images. It may be interesting to create more images in the tool, especially photorealistic ones, to perform a more thorough evaluation.
- **Changing colour spaces and interpolation functions after splitting:** The current splitting approach calculates the derivative values for

split patches at the time of splitting. Since these values must be updated to change the colour space or interpolation function, it is currently not possible to change these after splitting. To prevent this, the splitting approach must be adapted to dynamically update the derivatives of split patches at later times as well. This was also requested during the user study.

- **Other methods of influencing derivatives:** This research showed that assigning colours to derivatives can influence the colour gradient, but is no meaningful approach. However, users in the user study liked the added control over the colour gradient. Other approaches to controlling the gradient through the derivatives can be researched. Maybe it would be possible to implement a ‘lighten’ and ‘darken’ option at derivatives, and allow users to control the intensity of this effect?
- **More flexible influence on colours:** During the user study, several suggestions were made to allow more flexible control over colour in the gradient mesh. E.g., users asked for adding extra control points on edges and moving ‘colour’ control points without moving ‘position’ control points. While these may be difficult to implement in the gradient mesh, it would be interesting to research more flexible possibilities of influencing colour in the gradient mesh.
- **GUI improvements:** Many improvements to the GUI were requested during the user study. The most important one is an undo feature. All suggestions can be found in the results of the user study (Section 4.4).

Acknowledgements

I would like to thank my supervisors, Jiří Kosinka and Pieter Barendrecht, for their help and enthusiasm during this project. They were always willing to help me with questions or problems I encountered and were able to explain everything clearly, which I am grateful for.

Furthermore, I would like to thank the participants of the user study for their helpful feedback.

References

- [1] Adobe Systems Incorporated. Photoshop user guide: Convert between color modes. https://helpx.adobe.com/photoshop/using/converting-color-modes.html#convert_a_grayscale_or_rgb_image_to_indexed_color, 2017. Accessed: 2017-07-04.
- [2] Steven A. Coons. Surfaces for computer-aided design of space forms. Technical Report MIT/LCS/TR-41, MIT, 1967.
- [3] James Ferguson. Multivariable curve interpolation. *Journal of the ACM*, 11(2):221–228, 1964.
- [4] Pascal Getreuer. Converting color representations. <http://www.getreuer.info/home/colospace/>, 2010. Accessed: 2017-07-06.
- [5] The GIMP Documentation Team. GIMP user manual: Indexed mode. <https://docs.gimp.org/en/gimp-image-convert-indexed.html>, 2017. Accessed: 2017-07-04.
- [6] Yu-Kun Lai, Shi-Min Hu, and Ralph R. Martin. Automatic and topology-preserving gradient mesh generation for image vectorization. *ACM Transactions on Graphics*, 28(3), 2009.
- [7] Xian-Ying Li, Tao Ju, and Shi-Min Hu. Cubic mean value coordinates. *ACM Transactions on Graphics*, 32(4), 2013.
- [8] Henrik Lieng, Jiří Kosinka, Jingjing Shen, and Neil A. Dodgson. A colour interpolation scheme for topologically unrestricted gradient meshes. *Computer Graphics Forum*, 2016.
- [9] Martijn Luinstra. Locally refinable gradient meshes. Bachelor’s thesis, University of Groningen, 2017.
- [10] Mats Mattsson. Color interpolation. http://howaboutanorange.com/blog/2011/08/10/color_interpolation/, 2011. Accessed: 2017-07-04.
- [11] The Qt Company. Qt documentation: Qt concurrent. <http://doc.qt.io/qt-5/qtconcurrent-index.html>, 2017. Accessed: 2017-07-18.
- [12] Jian Sun, Lin Liang, Fang Wen, and Heung-Yeung Shum. Image vectorization using optimized gradient meshes. *ACM Transactions on Graphics*, 26(3), 2007.
- [13] Liang Wan, Yi Xiao, Ning Dou, Chi-Sing Leung, and Yu-Kun Lai. Scribble-based gradient mesh recoloring. *Multimedia Tools and Applications*, 2017.
- [14] Yi Xiao, Liang Wan, Chi-Sing Leung, Yu-Kun Lai, and Tien-Tsin Wong. Example-based color transfer for gradient meshes. *IEEE Transactions on Multimedia*, 15(3):549–560, 2013.